A

PROJECT REPORT ON

Implementation of Source Coding Techniques Using LabVIEW

SUBMITTED TO SAVITRIBAI PHULE PUNE UNIVERSITY
FOR PARTIAL FULFILLMENT
OF THE REQUIREMENTS
FOR THIRD YEAR

BACHELOR OF ENGINEERING
In
Electronics and Telecommunication Engineering

By

TEJAS BORA	**71828603M**
VAIBHAV THAKUR	**71829292J**
VYENKATESH PAREEK	**71829323B**

GUIDE
PROF. R.SHRIVAS

DEPARTMENT OF
ELECTRONICS AND TELECOMMUNICATION ENGINEERING
PUNE INSTITUTE OF COMPUTER TECHNOLOGY
PUNE – 43

Department of Electronics and Telecommunication Engineering
Pune Institute of Computer Technology, Pune – 43

CERTIFICATE

This is to certify that the project report entitled

Implementation of Source Coding Techniques using LabVIEW

has been successfully completed by

TEJAS BORA	71828603M
VAIBHAV THAKUR	71829292J
VYENKATESH PAREEK	71829323B

Is a bona fide work carried out by them towards the partial fulfillment of the requirement of the Savitribai Phule Pune University, Pune for the Third year ESMP subject of the Bachelor of Engineering in Electronics and Telecommunication Engineering. This project work has not been earlier submitted to any other Institute or University for the award of any degree or diploma.

Name	Dr. S.V. Gaikwad	Dr. P.T. Kulkarni
Project Guide	HOD, E&TC Dept	Principal, PICT

Place: Pune
Date :

ACKNOWLEDGEMENTS

Please acknowledge everybody who were involved and helped you in completing this report/theses/project work. But keep this brief and resist the temptation of writing flowery prose. Do include all those who helped you, e.g. other faculty / staff you consulted, colleagues who assisted etc. Acknowledge the source of any work that is not your own.

Tejas Bora

Vaibhav Thakur

Vyenkatesh Pareek

CONTENTS

CHAPTER 1

Introduction

1.1 Background

Coding theory is the study of the properties of codes and their respective fitness for specific applications. Codes are used for Coding theory is the study of the properties of codes and their respective fitness for specific applications. Codes are used for data compression, cryptography, error detection and correction, data transmission and data storage. Codes are studied by various scientific disciplines—such as information theory, electrical engineering, mathematics, linguistics, and computer science—for the purpose of designing efficient and reliable data transmission methods. This typically involves the removal of redundancy and the correction or detection of errors in the transmitted data.

There are four coding techniques

1. **Data compression (or source coding)**
2. **Error control (or channel coding)**
3. **Cryptographic coding**
4. **Line coding**

Data compression attempts to remove redundancy from the data from a source in order to transmit it more efficiently. For example, Zip data compression makes data files smaller, for purposes such as to reduce Internet traffic. Data compression and error correction may be studied in combination.

Error correction adds extra data bits to make the transmission of data more robust to disturbances present on the transmission channel. This typically involves the removal of redundancy and the correction or detection of errors in the transmitted data.

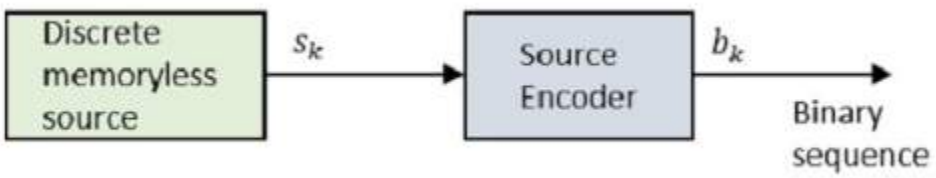
Data compression attempts to remove redundancy from the data from a source in order to transmit it more efficiently. Error correction adds extra data bits to make the transmission of data more robust to disturbances present on the transmission channel.

1.2 Relevance

The discrete memoryless source produces the code that has to be represented efficiently. It is one of the important problems in communications. There are code words representing these source codes.

Consider telegraphy where Morse code is used. The alphabets are represented by Marks and Spaces. If letter E is taken that is used mostly, it is denoted by "." Whereas the letter Q that is used rarely is denoted by "--.-"

Below is the block diagram.



Here $\mathbf{S_k}$ is the discrete memoryless source output and the b_k is the source encoder output which is represented by 0s and 1s.

The encoded sequence is easily decoded at the receiver.

Consider the source has an alphabet with k different symbols and the k^{th} symbol S_k occurs with the probability P_k, where k = 0, 1…k-1.

S_k is assigned with binary code work, by the encoder having length l_k, is measured in bits.

Hence, the average code word length L of the source encoder is defined as

$$\bar{L} = \sum_{k=0}^{k-1} p_k l_k$$

L represents the average number of bits per source symbol
If $\mathbf{L_{min}}$=minimum possible value of L
Then coding efficiency is defined as

$$\eta = \frac{Lmin}{\bar{L}}$$

With $\bar{L} \geq L_{min}$ we will have **η≤1**
Moreover, the source encoder is efficient when **η=1**
For this, the value $\mathbf{L_{min}}$ has to be determined.
The definition, "Given a discrete memoryless source of entropy **H(δ)**, the average code-word length **L** for any source encoding can bounded as $\bar{L} \geq H(\delta)$.
In simple terms, the code word (example: Morse code for the word QUEUE is -.- ..- . ..- .) is always greater than or equal to the source code (QUEUE in example). This means, code word symbols are greater than or equal to the source code alphabets.
Therefore with $\mathbf{L_{min}=H(\delta)}$, the source encoder efficiency in terms of Entropy **H(δ)** can be written as

$$\eta = \frac{H(\delta)}{\bar{L}}$$

This is known as noiseless coding theorem as it provides encoding which is error-free, also known as **Shannon's first** theorem.

1.3 Literature Survey

- S.Shanmugasundaram and R. Lourdusamy, "A Comparative Study of Text Compression Algorithms" There are lot of data compression algorithms which are available to compress files of different formats. This paper provides a survey of different basic lossless data compression algorithms. Experimental results and comparisons of the lossless compression algorithms using Statistical compression techniques and Dictionary based compression techniques were performed on text data [1].
- Md. RubaiyatHasan, "Data Compression using Huffman based LZW Encoding Technique" A method and system for transmitting a digital image (i.e., an array of pixels) from a digital data source to a digital data receiver. More the size of the data be smaller, it provides better transmission speed and saves time. In this communication we always want to transmit data efficiently and noise free [2].
- S. Kaur and V.S.Verma,"Design and Implementation of LZW Data Compression Algorithm "In this paper, LZW data compression algorithm is implemented by finite state machine, thus the text data can be effectively compressed [3].
- U.Khuranaand, A.Koul, "Text Compression And Superfast Searching" A new compression technique that uses referencing through two-byte numbers (indices) for the purpose of encoding has been presented. The technique is efficient in providing high compression ratios and faster search through the text [4].

1.4 Aim of the Project

Source coding (source compression coding) The use of variable-length codes in order to reduce the number of symbols in a message to the minimum necessary to represent the information in the message, or at least to go some way toward this, for a given size of alphabet. In source coding the particular code to be used is chosen to

match the source (i.e. the relative probabilities of the symbols in the source alphabet) rather than any channel through which the message may ultimately be passed.
The main problem in source coding is to ensure that the most probable source symbols are represented by the shortest codewords, and the less probable by longer codewords as necessary, the weighted average codeword length being minimized within the bounds of Kraft's inequality. The most widely used methods for ensuring this are Huffman coding and Shannon–Fano coding; the former is more efficient for a given extension of the source but the latter is computationally simpler. In either case, a large extension of the source may be necessary to approach the limiting compression factor given by the source coding theorem..

1.5 Scope and Objectives

This report provides a description of the fundamentals of source coding. It is aimed at aiding students and engineers to investigate the subject.,we chose to deal with it in greater depth. However, we make no attempt to exhaustive coverage of the subject, since it is too broad and too deep to fit the compact presentation format that is chosen here. Instead our focus is on the source coding fundamentals 1.2 Scope and Overview of the source coding. This means that we will leave out a number of areas including implementation aspects of source coding. We will be determining the following things through our project

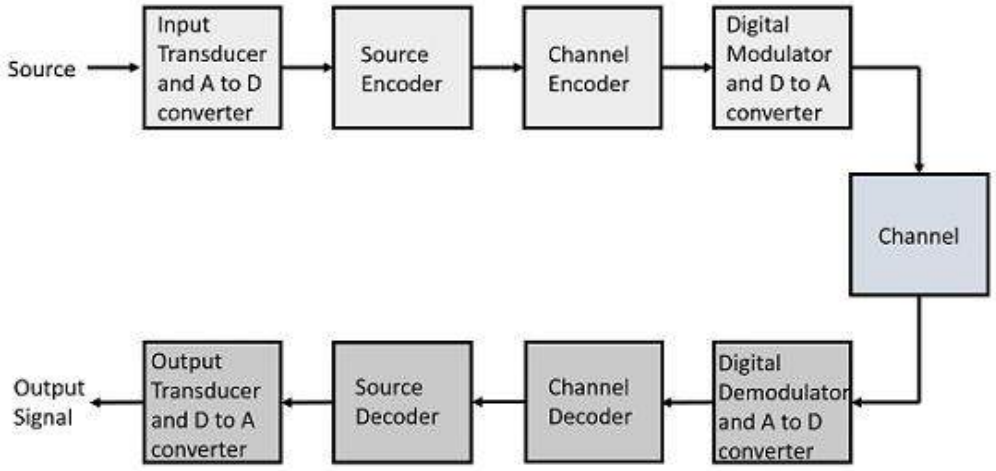
- Various source coding techniques and their algorithms
- Comparison of efficiency of all the source coding techniques

CHAPTER 2

Proposed System

2.1 Introduction

The advances in source coding technology along with the rapid developments and improvements of network infrastructures, storage capacity, and computing power are enabling an increasing number of multimedia applications. In this report, we will describe and analyze fundamental source coding techniques that are found in a variety of multimedia applications, with the emphasis on algorithms that are used in video coding applications. The block structure for a typical transmission scenario is illustrated in Figure 1.1.



Basic Elements of a Digital Communication System

The source generates a signal s. The source encoder maps the signal s into the bitstream b. The bitstream is transmitted over the error control channel and the received bitstream b is processed by the source decoder that reconstructs the decoded signal s and delivers it to the sink which is typically a human observer. This monograph focuses on the source encoder. The error characteristic of the digital channel can be controlled by the channel encoder, which adds redundancy to the bits at the source . The sequence of the five components, channel encoder, modulator, channel, demodulator, and channel decoder, are lumped into one box, which is called the error control channel. According to Shannon's basic work [63, 64] that also laid the ground to the subject of this text, by introducing redundancy at the channel encoder and by introducing delay, the amount of transmission errors can be controlled.

Digital media technologies have become an integral part of the way we create, communicate, and consume information. At the core of these technologies are source coding methods that are described in this report. Based on the fundamentals of information and rate distortion theory, the most relevant techniques used in source coding algorithms are described: shannon coding, hauffman coding and run length coding.

2.2 Shannon coding

In the field of data compression, Shannon coding, named after its creator, Claude Shannon, is a lossless data compression technique for constructing a prefix code based on a set of symbols and their probabilities (estimated or measured). It is suboptimal in the sense that it does not achieve the lowest possible expected code word length like Huffman coding does, and never better but sometimes equal to the Shannon–Fano coding.

The method was the first of its type, the technique was used to prove Shannon's noiseless coding theorem in his 1948 article "A Mathematical Theory of Communication", and is therefore a centerpiece of the information age. This coding method gave rise to the field of information theory and without its contribution, the world would not have any of the many successors; for example Shannon-Fano coding, Huffman coding, or arithmetic coding. Much of our day-to-day lives are significantly influenced by digital data and this would not be possible without Shannon coding and its ongoing evolution of its predecessor coding methods.

2.3 Huffman Coding

In computer science and information theory, a **Huffman code** is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding or using such a code proceeds by means of **Huffman coding**, an algorithm developed by **David A. Huffman** while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (*weight*) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted. However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods - it is replaced with arithmetic coding or asymmetric numeral systems if better compression ratio is required.
Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix code (sometimes called "prefix-free codes", that is, the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol). Huffman coding is such a widespread method for creating prefix codes that the term "Huffman code" is widely used as a synonym for "prefix code" even when such a code is not produced by Huffman's algorithm.

2.4 Run Length Coding

Run-length encoding (**RLE**) is a form of lossless data compression in which *runs* of data (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs. Consider, for example, simple graphic images such as icons, line drawings, Conway's Game of Life, and animations. It is not useful with files that don't have many runs as it could greatly increase the file size.
RLE may also be used to refer to an early graphics file format supported by CompuServe for compressing black and white images, but was widely supplanted by their later Graphics Interchange Format (GIF). RLE also refers to a little-used image format in Windows 3.x, with the extension rle, which is a Run Length Encoded Bitmap, used to compress the Windows 3.x startup screen.

CHAPTER 3

Implementation and Working Principe

Shannon Coding

In Shannon–Fano coding, the symbols are arranged in order from most probable to least probable, and then divided into two sets whose total probabilities are as close as possible to being equal. All symbols then have the first digits of their codes assigned; symbols in the first set receive "0" and symbols in the second set receive "1". As long as any sets with more than one member remain, the same process is repeated on those sets, to determine successive digits of their codes. When a set has been reduced to one symbol, of course, this means the symbol's code is complete and will not form the prefix of any other symbol's code. The algorithm works, and it produces fairly efficient variable-length encodings; when the two smaller sets produced by a partitioning are in fact of equal probability, the one bit of information used to distinguish them is used most efficiently. Unfortunately, Shannon–Fano does not always produce optimal prefix codes. For this reason, Shannon–Fano is almost never used; Huffman coding is almost as computationally simple and produces prefix codes that always achieve the lowest expected code word length. Shannon–Fano coding is used in the IMPLODE compression method, which is part of the ZIP file format, where it is desired to apply a simple algorithm with high performance and minimum requirements for programming.

Huffman Coding

The technique works by creating a binary tree of nodes. These can be stored in a regular array, the size of which depends on the number of symbols, A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the **symbol** itself, the **weight** (frequency of appearance) of the symbol and optionally, a link to a **parent** node which makes it easy to read the code (in reverse) starting from a leaf node. Internal nodes contain a **weight**, links to **two child nodes** and an optional link to a **parent** node. As a common convention, bit '0' represents following the left child and bit '1' represents following the right child. A finished tree has up to leaf nodes and internal nodes. A Huffman tree that omits unused symbols produces the most optimal code lengths.
The process begins with the leaf nodes containing the probabilities of the symbol they represent. Then, the process takes the two nodes with smallest probability, and creates a new internal node having these two nodes as children. The weight of the new node is set to the sum of the weight of the children. We then apply the process again, on the new internal node and on the remaining nodes (i.e., we exclude the two leaf nodes), we repeat this process until only one node remains, which is the root of the Huffman tree.
The simplest construction algorithm uses a priority queue where the node with lowest probability is given highest priority:
Create a leaf node for each symbol and add it to the priority queue.
While there is more than one node in the queue:
Remove the two nodes of highest priority (lowest probability) from the queue
Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
Add the new node to the queue.

The remaining node is the root node and the tree is complete.
Since efficient priority queue data structures require O(log n) time per insertion, and a tree with n leaves has $2n-1$ nodes, this algorithm operates in O(n log n) time, where n is the number of symbols.
If the symbols are sorted by probability, there is a linear-time (O(n)) method to create a Huffman tree using two queues, the first one containing the initial weights (along with pointers to the associated leaves), and combined weights (along with pointers to the trees) being put in the back of the second queue. This assures that the lowest weight is always kept at the front of one of the two queues:
Start with as many leaves as there are symbols.
Enqueue all leaf nodes into the first queue (by probability in increasing order so that the least likely item is in the head of the queue).
While there is more than one node in the queues:
Dequeue the two nodes with the lowest weight by examining the fronts of both queues.
Create a new internal node, with the two just-removed nodes as children (either node can be either child) and the sum of their weights as the new weight.
Enqueue the new node into the rear of the second queue.
The remaining node is the root node; the tree has now been generated.
In many cases, time complexity is not very important in the choice of algorithm here, since n here is the number of symbols in the alphabet, which is typically a very small number (compared to the length of the message to be encoded); whereas complexity analysis concerns the behavior when n grows to be very large.
It is generally beneficial to minimize the variance of codeword length. For example, a communication buffer receiving Huffman-encoded data may need to be larger to deal with especially long symbols if the tree is especially unbalanced. To minimize variance, simply break ties between queues by choosing the item in the first queue. This modification will retain the mathematical optimality of the Huffman coding while both minimizing variance and minimizing the length of the longest character code.

Run Length Coding

Run-length encoding (RLE) is one of the simplest data compression methods. Consider the example in which we have represented an MxN image whose top half s totally white, and bottom half is totally black. That example was a primitive attempt to encode the image using RLE. The principle of RLE is to exploit the repeating values in a source. The algorithm counts the consecutive repetition amount of a symbol and uses that value to represent the run. This simple principle works best on certain source types in which repeated data values are significant. Black-White document images, cartoon images, etc. are quite suitable for RLE. Actually, RLE may be used on any kind of source regardless of its content, but its compression efficiency changes significantly depending on the above types of data are used or not. As another application suitable for RLE, we can mention text files which contains multiple spaces for indention and formatting paragraphs, tables and charts. Principle : As indicated above, basic RLE principle is that the run of characters is replaced with the number of the same characters and a single character. Examples may be helpful to understand it better. Ex. 8: Consider a text source: R T A A A A S D E E E E E The RLE representation is: R T *4A S D *5E This example also shows how to distinguish whether a symbol corresponds to the daa value or its repetition count (called run). Each repeating bunch of characters is replaced with three symbols: an indicator (*), number of characters, and the character itself. We need the indication of he redundant character * to separate between the encoding of a repeating cluster and a single character. In the above example, if there is no repetition around a character, it is encoded as itself. In the above example, it is important to realize that the encoding process is effective only if there are sequences of 4 or more repeating characters because three characters are used to conduct RLE. For example, coding two repeating characters would lead to expansion and coding three repeating characters wouldn't cause compression or expansion since we represent a repetitive cluster with at least three symbols. The decoding process is easy: If there aren't control characters (*) the coded symbol just corresponds to the original symbol, and if control character occurred then it must be replaced with characters in a defined number of times. It can be noticed that the process of decoding control characters don't lead to any special procedures. The RLE symbolism and details

are not unique. There are many different run-length encoding schemes. The above example has just been used to demonstrate the basic principle of RLE encoding. In a particular case the implementation of RLE depends on what type of data is being compressed.

CHAPTER 4

Algorithms and Flowcharts

Shannon Algorithm:

A Shannon–Fano tree is built according to a specification designed to define an effective code table. The actual algorithm is simple:
1. For a given list of symbols, develop a corresponding list of probabilities or frequency counts so that each symbol's relative frequency of occurrence is known.
2. Sort the lists of symbols according to frequency, with the most frequently occurring symbols at the left and the least common at the right.
3. Divide the list into two parts, with the total frequency counts of the left part being as close to the total of the right as possible.
4. The left part of the list is assigned the binary digit 0, and the right part is assigned the digit 1. This means that the codes for the symbols in the first part will all start with 0, and the codes in the second part will all start with 1.
5. Recursively apply the steps 3 and 4 to each of the two halves, subdividing groups and adding bits to the codes until each symbol has become a corresponding code leaf on the tree

Huffman Algorithm:

1.Calculate the frequency of each character in the string.

2.Sort the characters in increasing order of the frequency. These are stored in a priority

3.Make each unique character as a leaf node.

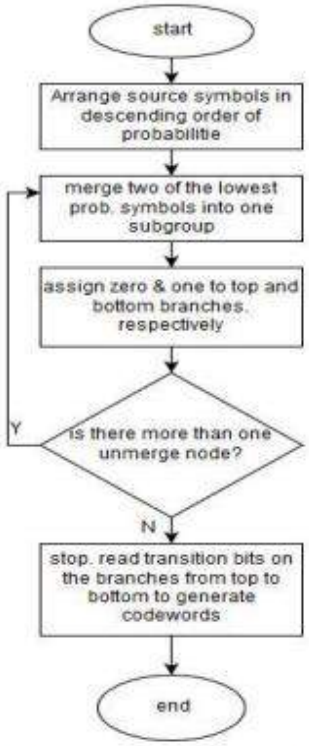
4.Create an empty node z. Assign the minimum frequency to the left child of z and assign the second minimum frequency to the right child of z. Set the value of the z as the sum of the above two minimum frequencies.

5.Remove these two minimum frequencies from Q and add the sum into the list of frequencies (* denote the internal nodes in the figure above).

6.Insert node z into the tree.

7.Repeat steps 3 to 5 for all the characters.

8.For each non-leaf node, assign 0 to the left edge and 1 to the right edge.



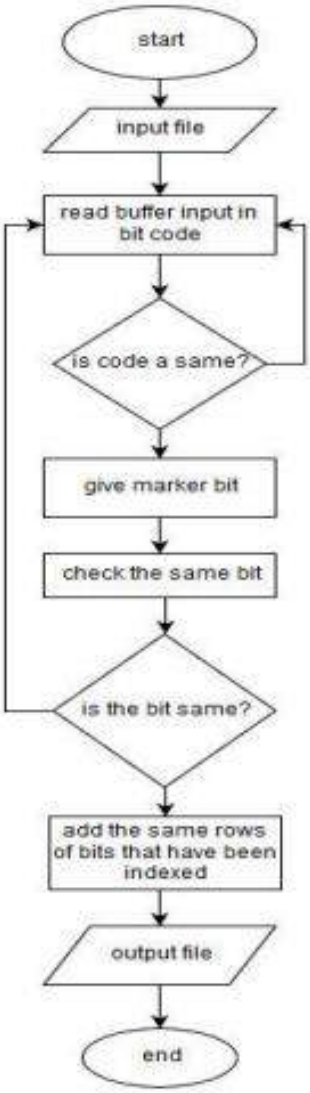
Run Length Coding:

1) Pick the first character from source string.

2) Append the picked character to the destination string.

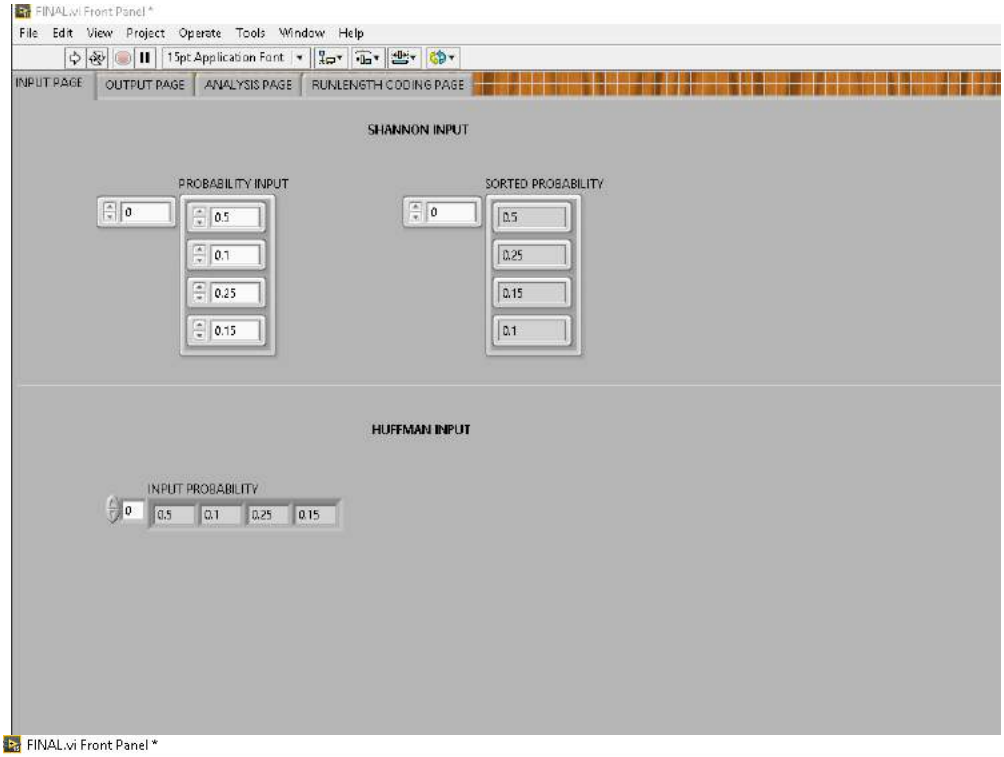
3) Count the number of subsequent occurrences of the picked character and append the count to destination string.

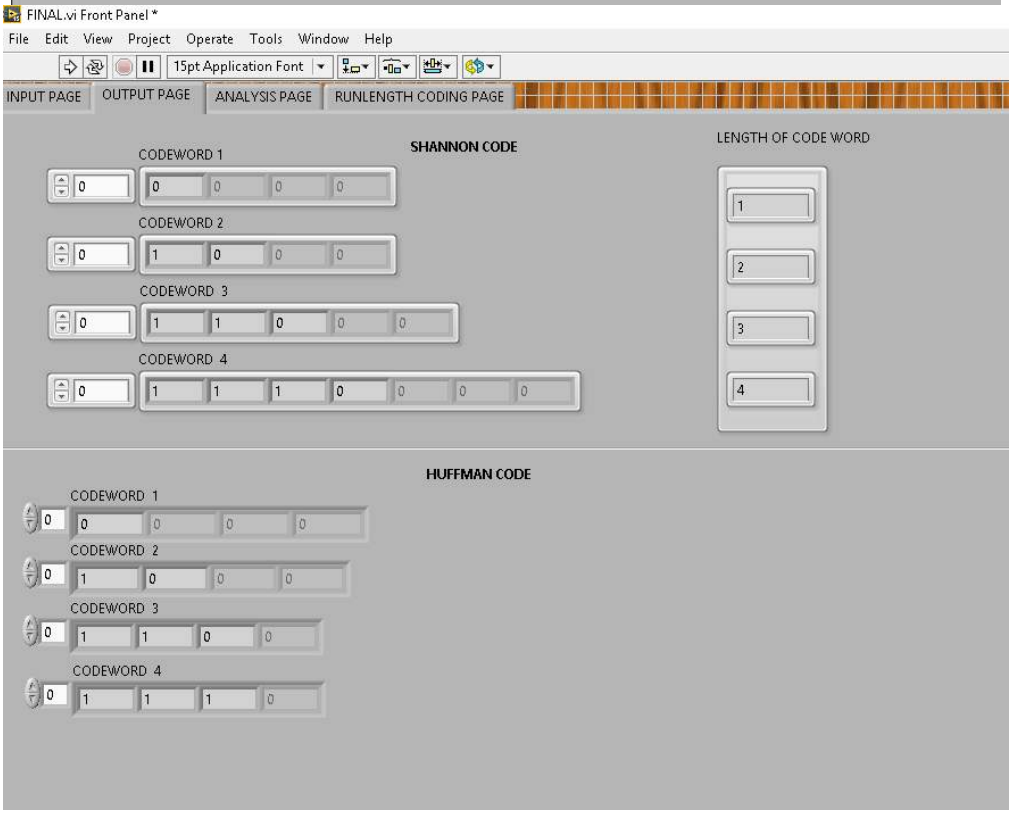
4) Pick the next character and repeat steps 2) 3) and 4) if end of string is NOT reached.

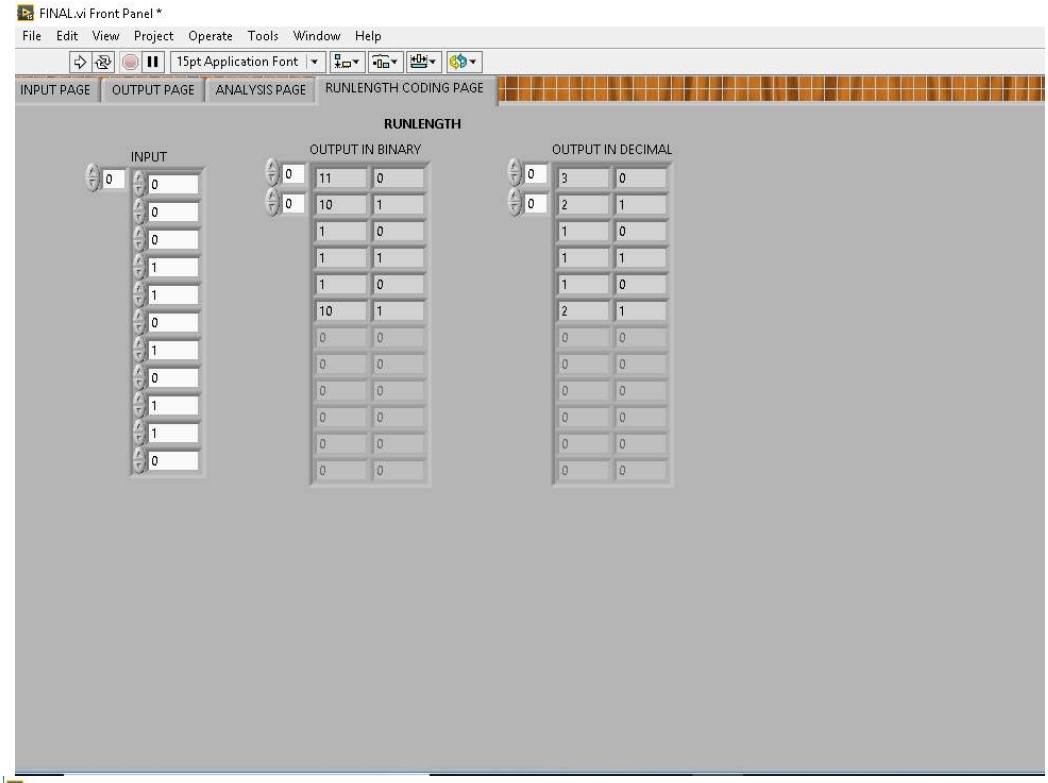

start
input file
read buffer input in bit code
is code a same?
give marker bit
check the same bit
is the bit same?
add the same rows of bits that have been indexed
output file
end

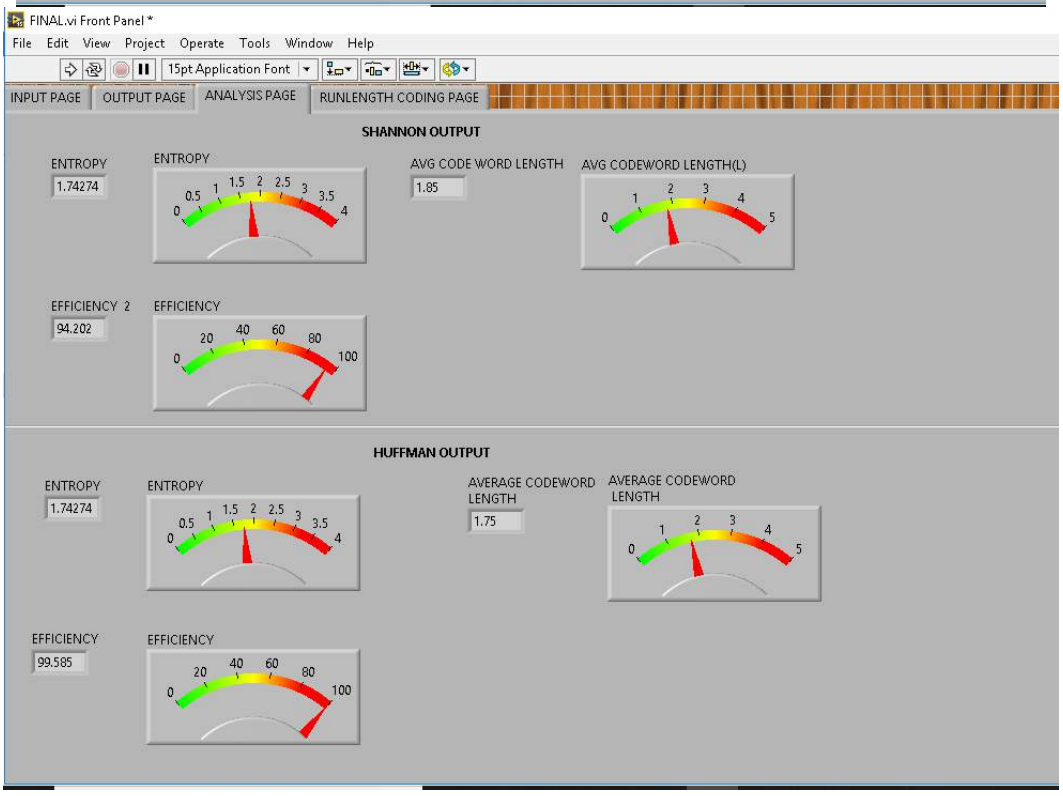
Chapter 5

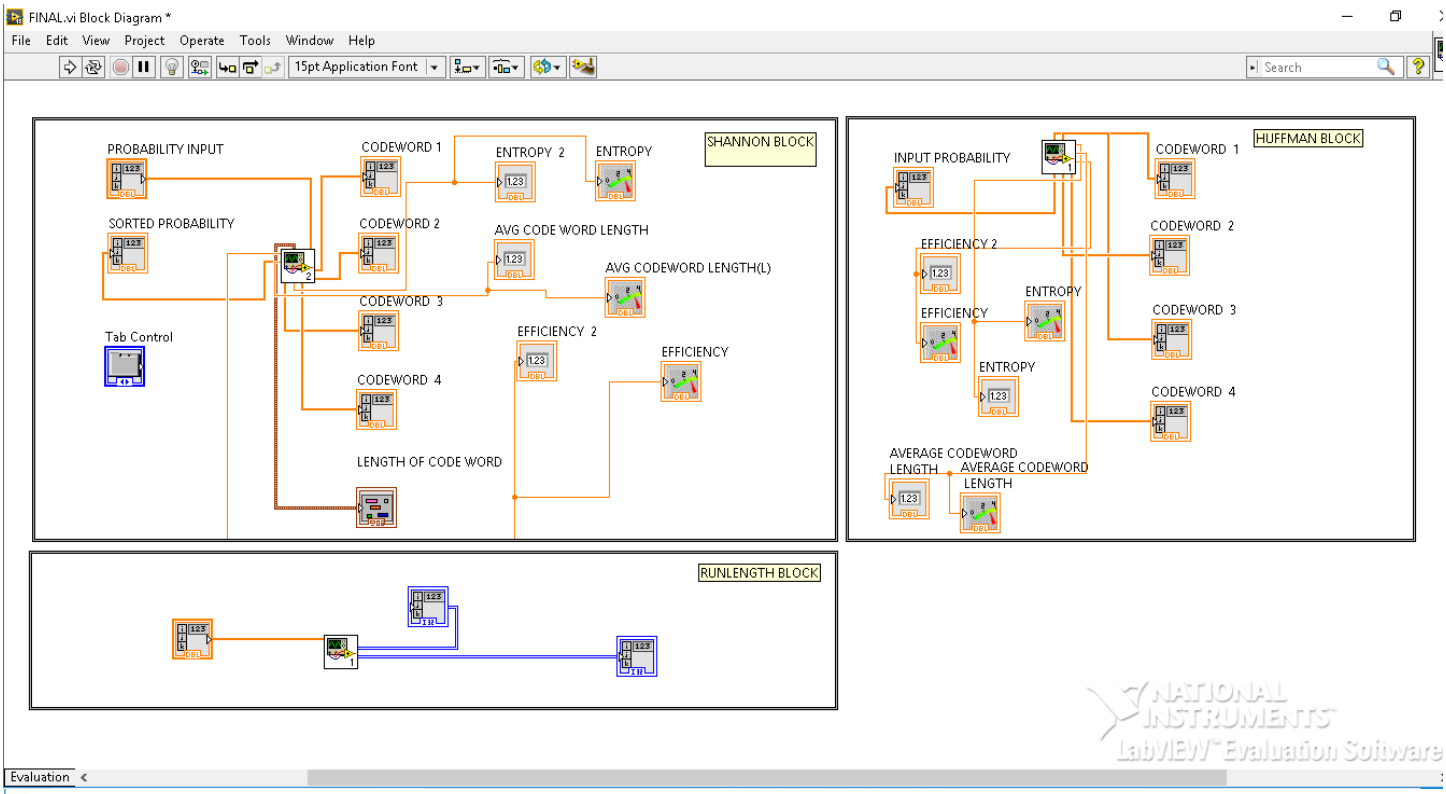
Results and Discussion

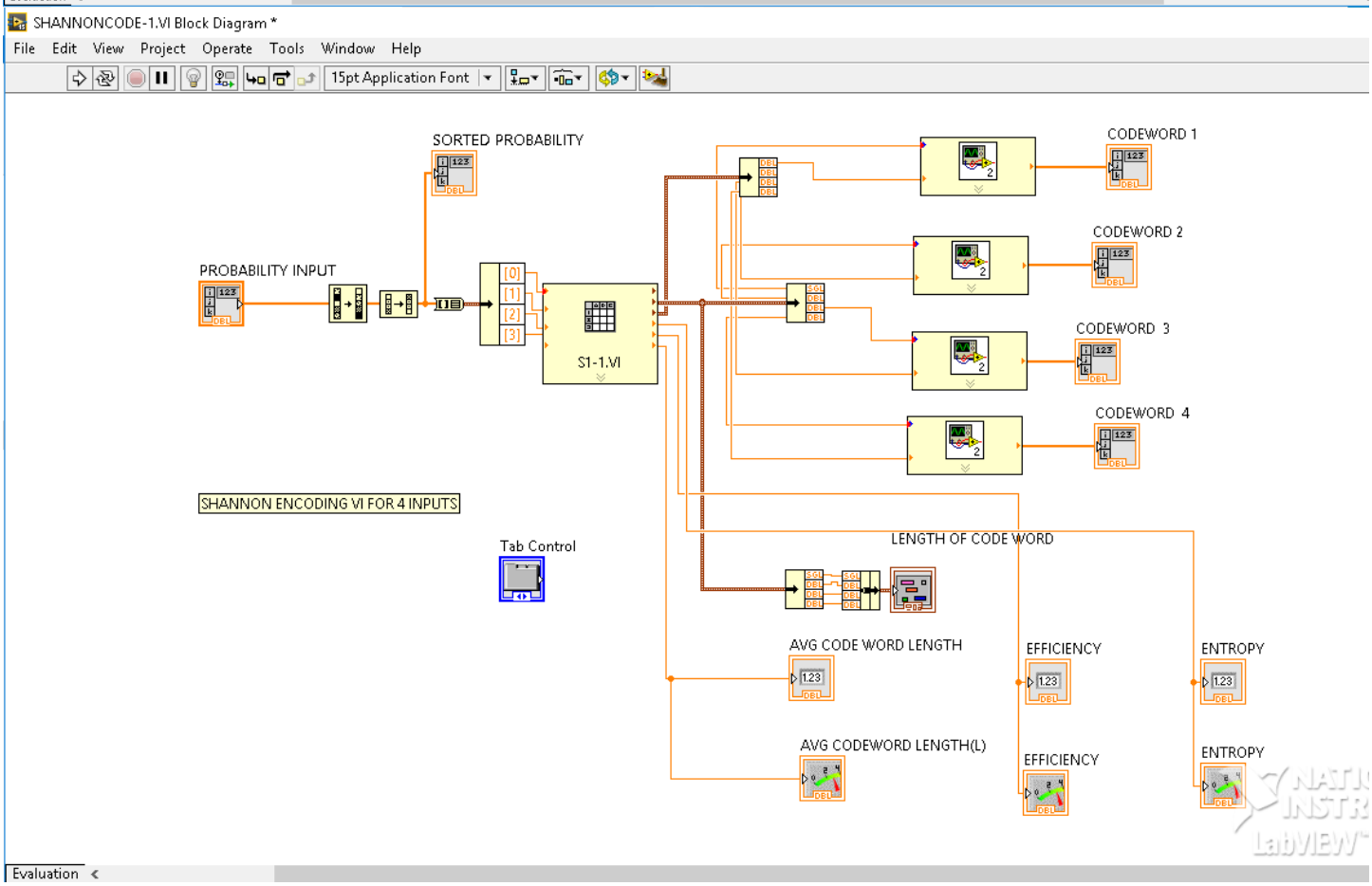


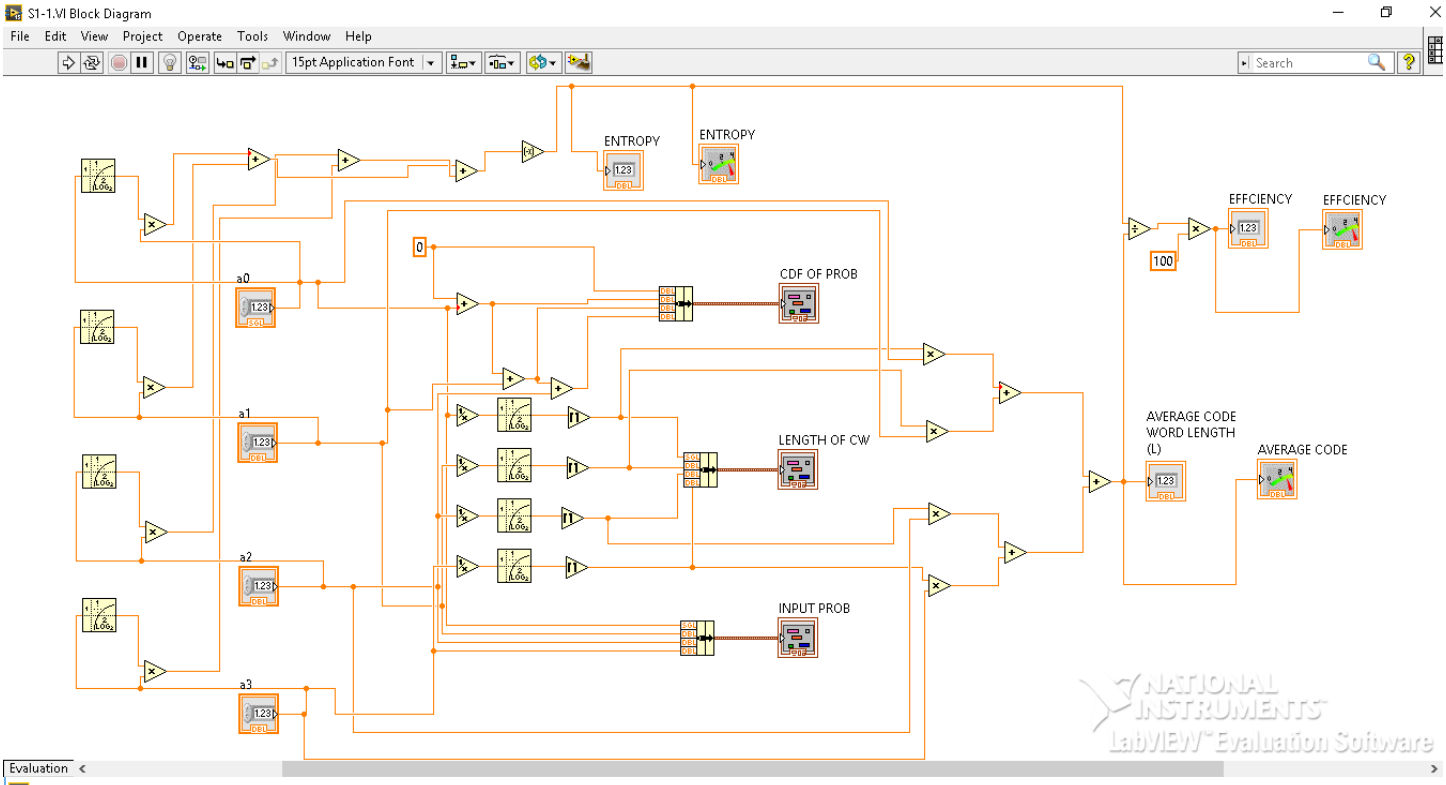


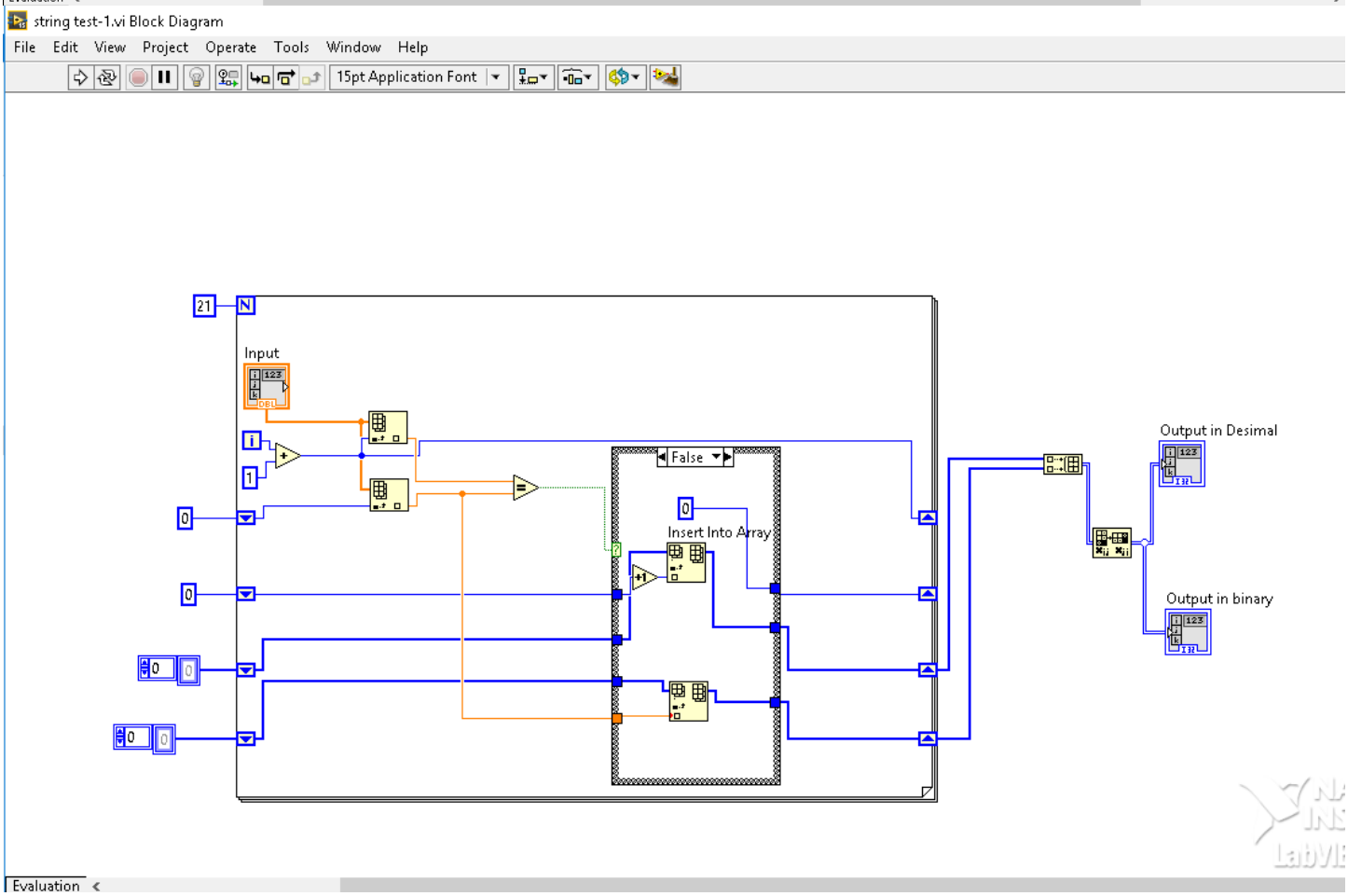

FINAL.vi Front Panel *
File Edit View Project Operate Tools Window Help
15pt Application Font
INPUT PAGE
OUTPUT PAGE
ANALYSIS PAGE
RUNLENGTH CODING PAGE
RUNLENGTH
INPUT
OUTPUT IN BINARY
OUTPUT IN DECIMAL


FINAL.vi Front Panel *
File Edit View Project Operate Tools Window Help
15pt Application Font
INPUT PAGE
OUTPUT PAGE
ANALYSIS PAGE
RUNLENGTH CODING PAGE
SHANNON OUTPUT
ENTROPY
1.74274
ENTROPY
AVG CODE WORD LENGTH
1.85
AVG CODEWORD LENGTH(L)
EFFICIENCY 2
94.202
EFFICIENCY
HUFFMAN OUTPUT
ENTROPY
1.74274
ENTROPY
AVERAGE CODEWORD LENGTH
1.75
AVERAGE CODEWORD LENGTH
EFFICIENCY
99.585
EFFICIENCY


FINAL.vi Block Diagram *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
SHANNON BLOCK
PROBABILITY INPUT
SORTED PROBABILITY
Tab Control
CODEWORD 1
CODEWORD 2
CODEWORD 3
CODEWORD 4
ENTROPY 2
ENTROPY
AVG CODE WORD LENGTH
AVG CODEWORD LENGTH(L)
EFFICIENCY 2
EFFICIENCY
LENGTH OF CODE WORD
HUFFMAN BLOCK
INPUT PROBABILITY
CODEWORD 1
CODEWORD 2
CODEWORD 3
CODEWORD 4
EFFICIENCY 2
EFFICIENCY
ENTROPY
ENTROPY
AVERAGE CODEWORD LENGTH
AVERAGE CODEWORD LENGTH
RUNLENGTH BLOCK
NATIONAL INSTRUMENTS
LabVIEW Evaluation Software
Evaluation


SHANNONCODE-1.VI Block Diagram *
File Edit View Project Operate Tools Window Help
15pt Application Font
SORTED PROBABILITY
PROBABILITY INPUT
[0]
[1]
[2]
[3]
S1-1.VI
CODEWORD 1
CODEWORD 2
CODEWORD 3
CODEWORD 4
SHANNON ENCODING VI FOR 4 INPUTS
Tab Control
LENGTH OF CODE WORD
AVG CODE WORD LENGTH
AVG CODEWORD LENGTH(L)
EFFICIENCY
EFFICIENCY
ENTROPY
ENTROPY
NATIONAL INSTRUMENTS
LabVIEW
Evaluation


S1-1.VI Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
ENTROPY
ENTROPY
EFFCIENCY
EFFCIENCY
100
CDF OF PROB
a0
a1
a2
a3
LENGTH OF CW
AVERAGE CODE WORD LENGTH (L)
AVERAGE CODE
INPUT PROB
NATIONAL INSTRUMENTS
LabVIEW Evaluation Software
Evaluation


string test-1.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
21
N
Input
1
1
0
0
0
0
False
Insert Into Array
+1
Output in Desimal
Output in binary
Evaluation

Chapter 6

Conclusions

After doing the experiments above, it can be concluded that, in Shannon-Fano, Huffman algorithm always get smaller result than previous bit. As for the Run Length Encoding algorithm, it depends on the sentence used. If the sentence used is a sentence that has a meaning, usually the algorithm is always obtained results greater than the previous bit. Whereas if the word used is a word that has a loop, it can produce a smaller end result than the previous bit. So it all depends on the data used. Similarly, the Compression Ratio, the percent value obtained by the Shannon-Fano algorithm, the Huffman algorithm, always have a high percentage or high enough value. Unlike the Run Length Encoding algorithm, the algorithm if the sentence contains meaning, always get a relatively small percentage. Because the result of the calculation is greater than the initial bit value. Thus affecting the percent value of the compression ratio.

Chapter 7

Future Scope

As a future work more focus can be on improvement of compression ratio using the new techniques. The proposed technique can be experimented on different kinds of data sets like audio, video, text as till now it is restricted to images. New methods can be combined and proposed that decreases the time complexity incurred in creating dictionary. The experimental dataset in this research is somehow limited; so applying the developed methods on a larger dataset could be a subject for future research which may lead to new observations and conclusions. Further the work can be extended to video compression. Video data is basically a three dimensional array of color pixels, that contains spatial and temporal redundancy.

References

International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 23 (2017) pp. 13618-13622 © Research India Publications. http://www.ripublication.com

Ramkrishna Rao,Digital Communication reference books

http://web.stanford.edu>informationtheory

Class Notes